Hacettepe University

Department of Artificial Intelligence Engineering

Battle of Ships Assignment Report

Course Name:

BBM103 – Introduction to Programming Lab

Assignment 4

Report Prepared by:

Özge Bülbül

2220765008

January 3, 2022

Table of Contents

1 Introduction

Battle of Ships is a two-player, competitive game played on a 10 x 10 surface where each individual player has a set of ships and try to sink their opponent's ships. Each square on the surface is defined by a letter and a number. The ships are shown with their information in the figure below.

No.	Class of ship	Size	Count	Label
1	Carrier	5	1	CCCCC
2	Battleship	4	2	BBBB
3	Destroyer	3	1	DDD
4	Submarine	3	1	SSS
5	Patrol Boat	2	4	PP

Players take turns and shoot one square of the opponents 10 x 10 plane. The shot square is shown as "X" if the player shot successfully and as "O" if not. When all the ships of one player are sunk by the opponent, the opponent wins the game. If all ships are sunk by the end of the round, it's a draw.

2 Analysis

The game is based on a round system and each player makes a move each round. Both of the players' ship positions and moves are taken from separate input files. The patrol ships and battleships are in multiple amounts. Therefore, they require separate input files to take their positions with certainty. The result of the game with all rounds shown in detail and all the errors faced during the playtime are written in an output file and also printed to the command line.

There is a quite various set of things that may go wrong during the implementation of this game. For that, many exceptions are handled to avoid the occurrence of unwanted situations.

3 Design

3.1 Reading and Writing the Data

In the Battle of Ships, the data is supposed to be taken from and written to a .txt file. Four of the input files are taken as argument, these are players' moves and ship placements. Two other input files are not taken as argument and they are the optional files that contain placements of the ships that are in multiple amounts. This is applied by importing os and sys module using the following part of the code.

```
import sys
import os
current_dir_path = os.getcwd()
reading_player1_ship = "OptionalPlayer1.txt"
reading_file_path1 = os.path.join(current_dir_path, reading_player1_ship)
reading_player2_ship = "OptionalPlayer2.txt"
reading_file_path2 = os.path.join(current_dir_path, reading_player2_ship)
writing_file_name = "Battleship.out"
writing_file_path = os.path.join(current_dir_path, writing_file_name)
```

3.2 The Write Function

The code segment given below is the writing output function that writes the output to our output file and also to the command line.

```
def write(output):
    with open(writing_file_path, 'a') as f:
        f.write(output)
        print(output, end='')
```

3.3 The Move Checker Function

The move checker function checks if every move given in move txt files are correct and valid. If the move is missing characters, it raises index error. If the move given is not correct like two letters ("A,A;"), or two numbers ("2,4;") it raises value error. Finally, if the move given is not in the limits of our 10x10 board like the numbers above 10 and letters after J, it raises assertion error. This code segment is given below.

```
        def move_checker(x):
    for move in x:
        try:
            if len(move) < 3:
                raise IndexError
            if type(int(move[:-2])) != int or type(move[-1]) != str:
                raise ValueError
            if int(move[:-2]) > 10 or ord(move[-1]) > 74:
                raise AssertionError
        except IndexError:
            write("IndexError: Move value {} is
insufficient.\n".format(move))
            moves_p1.remove(move)
```

```
        except ValueError:
            write("ValueError: Move value {} is invalid.\n".format(move))
            moves_p1.remove(move)
        except AssertionError:
            write("AssertionError: Move value {} is not
possible.\n".format(move))
            moves_p1.remove(move)
```

3.4 The Sunk Ship Checker Function

The sunk ship checker function checks if the ship is sunk or not. This function is mainly defined to be able to give the following output.

```
Carrier    X                  Carrier    X
Battleship X X                Battleship - -
Destroyer  X                  Destroyer  -
Submarine  X                  Submarine  X
Patrol BoatX X X X            Patrol BoatX X - -
```

Here, you may see the first player's half of the function.

```
def sunk_ship_checker():
    c_counter = 0
    s_counter = 0
    d_counter = 0
    general_p_counter = 0
    general_b_counter = 0
    b1_counter = 0
    b2_counter = 0
    p1_counter = 0
    p2_counter = 0
    p3_counter = 0
    p4_counter = 0
    for element in sunk1:
        element1 = element.split(",")
        type_ship = element1[0]
        if type_ship == "C":
            c_counter += 1
        if type_ship == "S":
            s_counter += 1
        if type_ship == "D":
            d_counter += 1
        if type_ship == "B1":
            b1_counter += 1
        if type_ship == "B2":
            b2_counter += 1
        if type_ship == "P1":
            p1_counter += 1
        if type_ship == "P2":
            p2_counter += 1
        if type_ship == "P3":
            p3_counter += 1
        if type_ship == "P4":
            p4_counter += 1
    if c_counter == 5:
```

```
        global c
        c = "X"
    if s_counter == 3:
        global sub
        sub = "X"
    if d_counter == 3:
        global d
        d = "X"
    if p1_counter == 2:
        general_p_counter += 1
    if p2_counter == 2:
        general_p_counter += 1
    if p3_counter == 2:
        general_p_counter += 1
    if p4_counter == 2:
        general_p_counter += 1
    if general_p_counter == 4:
        global p
        p = " X X X X"
    elif general_p_counter == 3:
        p = " X X X -"
    elif general_p_counter == 2:
        p = " X X - -"
    elif general_p_counter == 1:
        p = " X - - -"
    if b1_counter == 4:
        general_b_counter += 1
    if b2_counter == 4:
        general_b_counter += 1
    if general_b_counter == 2:
        global b
        b = "X X"
    elif general_b_counter == 1:
        b = "X -"
```

3.5 The Hidden Board Function

The hidden board function creates the hidden board output that may look like this.

```
Player1's Hidden Board      Player2's Hidden Board
  A B C D E F G H I J         A B C D E F G H I J
1 O - - O - O X - O O       1 - - - - - - - - - -
2 - - O - - - X - O -       2 - - X - - X X O O X
3 - - - - - - O - - - -     3 - - - - - - - O - - X
4 O - - - - - - X O - O     4 X O X - O O O - - -
5 O O O - - O - - - O       5 - - - - - O O X - - -
6 - X X X X - O - O -       6 O - O - - - O O - -
7 O O - - - - - - - - O     7 O - - O - - - - - -
8 O - - O O O - - O X       8 - - - - - - O O X O -
9 - - - - - - - O - - -     9 - - O - X - - - - -
10- - - O O - - - - O -     10O - O O O - O O O -
```

In this function, you may see that every cell is marked as "-". Once the cell is hit, if it is empty it looks like this: "O". If there was a ship section in there, it looks like this: "X".

Since 10 is a two-digit number, a small exception is made for the table id the row is equal to 10.

```
def hidden_board():
    sunk_ship_checker()
    global game
    global check
    if c == "X" and sub == "X" and d == "X" and p == " X X X X" and b == "X
X" and c2 == "X" and sub2 == "X" and d2 == "X" and p_2 == " X X X X" and
b_2 == "X X" and not check:
        write("It is a Draw!\n\nFinal Information\n\n")
        check = True
        game = "over"
        return
    if c == "X" and sub == "X" and d == "X" and p == " X X X X" and b == "X
X" and not check:
        write("Player2 Wins!\n\nFinal Information\n\n")
        check = True
        game = "over"
        return
    if c2 == "X" and sub2 == "X" and d2 == "X" and p_2 == " X X X X" and
b_2 == "X X" and not check:
        write("Player1 Wins!\n\nFinal Information\n\n")
        check = True
        game = "over"
        return
    write("Player1's Hidden Board\t\tPlayer2's Hidden Board\n  A B C D E F
G H I J\t\t  A B C D E F G H I J\n")
    check = False
    if tarantino:
        a = round_num
        if round_num == len(moves_p1) + 1:
            move = moves_p2[round_num - 2]
        else:
            move = moves_p2[round_num - 1]
    else:
        a = round_num - 1
        if round_num == len(moves_p1) + 1:
            move = moves_p2[round_num - 2]
        else:
            move = moves_p2[round_num - 1]
    for row in range(10):
        write(str(row + 1))
        for column in range(10):
            x = False
            for number in range(round_num - 1):
                let = moves_p2[number][-1]
                num1 = moves_p2[number][:-2]
                if row == 9 and column + 1 == ord(let) - 64 and row + 1 ==
int(num1):
                    for s in first_ships:
                        s1 = s.split(",")
                        position = s1[1]
                        if position[-1] == let and position[:-1] == num1:
                            check = True
                            if s not in sunk1:
                                sunk1.append(s)
                    if check:
                        write("X ")
                        x = True
                        check = False
                    else:
                        write("O ")
                        x = True
```

```
                elif column + 1 == ord(let) - 64 and row + 1 == int(num1):
                    for s in first_ships:
                        if s[-1] == let and s[-2] == num1:
                            check = True
                            if s not in sunk1:
                                sunk1.append(s)
                    if check:
                        write(" X")
                        x = True
                        check = False
                    else:
                        write(" O")
                        x = True
            if row == 9 and not x:
                write("- ")
            elif not x:
                write(" -")
        write("\t\t")
        write(str(row + 1))
        for column in range(10):
            y = False
            for number in range(a):
                let2 = moves_p1[number][-1]
                num2 = moves_p1[number][:-2]
                if row == 9 and column + 1 == ord(let2) - 64 and row + 1 ==
int(num2):
                    for s in second_ships:
                        s1 = s.split(",")
                        position = s1[1]
                        if position[-1] == let2 and position[:-1] == num2:
                            check = True
                            if s not in sunk2:
                                sunk2.append(s)
                    if check:
                        write("X ")
                        y = True
                        check = False
                    else:
                        write("O ")
                        y = True
                elif column + 1 == ord(let2) - 64 and row + 1 == int(num2):
                    for s in second_ships:
                        if s[-1] == let2 and s[-2] == num2:
                            check = True
                            if s not in sunk2:
                                sunk2.append(s)
                    if check:
                        write(" X")
                        y = True
                        check = False
                    else:
                        write(" O")
                        y = True
            if row == 9 and not y:
                write("- ")
            elif not y:
                write(" -")
        write("\n")

write("\nCarrier\t\t{}\t\t\t\tCarrier\t\t{}\nBattleship\t{}\t\t\t\tBattlesh
ip\t{}\nDestroyer\t{}\t\t\t\t"
```

```
            "Destroyer\t{}\nSubmarine\t{}\t\t\t\tSubmarine\t{}\nPatrol
Boat{}\t\t\tPatrol Boat{}\n\n"
            .format(c, c2, b, b_2, d, d2, sub, sub2, p, p_2))
    if game != "over":
        write("Enter your move: {}\n\n".format(move))
```

3.6 Implementing the Code

Here, “Battle of Ships Game” is written at the start and then each round, the hidden board function is called twice. The “tarantino” variable is defined to make sure each round the first player makes their move first and then the second player. This code segment is given below.

```
write("Battle of Ships Game\n\n")
for round_num in range(1, len(moves_p1) + 2):
    if game != "over":
        write("Player1's Move\n\nRound : {}\t\t\t\t\tGrid Size:
10x10\n\n".format(round_num))
        tarantino = False
        hidden_board()
        tarantino = True
        write("Player2's Move\n\nRound : {}\t\t\t\t\tGrid Size:
10x10\n\n".format(round_num))
        hidden_board()
if game == "over":
    check = True
    round_num = len(moves_p1)
    hidden_board()
```

4 Programmer’s Catalogue

In this section, the whole code is provided and then time spent on analyzing, designing, implementing, testing and reporting and the reusability of the code are explained.

4.1 The Code

```
# Özge Bülbül 2220765008
# import sys
import sys
import os
current_dir_path = os.getcwd()
reading_player1_ship = "OptionalPlayer1.txt"
reading_file_path1 = os.path.join(current_dir_path, reading_player1_ship)
reading_player2_ship = "OptionalPlayer2.txt"
reading_file_path2 = os.path.join(current_dir_path, reading_player2_ship)
writing_file_name = "Battleship.out"
writing_file_path = os.path.join(current_dir_path, writing_file_name)
moves_p1 = []
moves_p2 = []
p1_ships = []
p2_ships = []
bp_positions_p1 = []
```

```
bp_positions_p2 = []
game = "on"


def write(output):
    print(output, end='')
    with open(writing_file_path, 'a') as f:
        f.write(output)


for i in range(1, 5):
    try:
        if i == 1:
            with open(sys.argv[i], 'r') as file:
                for n in file:
                    line5 = n.strip("\n")
                    line5 = line5.split(";")
                    p1_ships.append(line5)
        if i == 2:
            with open(sys.argv[i], 'r') as file:
                for n in file:
                    line5 = n.strip("\n")
                    line5 = line5.split(";")
                    p2_ships.append(line5)
        if i == 3:
            with open(sys.argv[i], 'r') as file:
                for n in file:
                    lines = n[:-1]
                    moves_p1.extend(lines.split(";"))
        if i == 4:
            with open(sys.argv[i], 'r') as file:
                for n in file:
                    lines = n[:-1]
                    moves_p2.extend(lines.split(";"))

    except IOError:
        write("IOError: input file(s) {} is/are not
reachable.".format(sys.argv[i]))


with open(reading_file_path1, 'r') as reading:
    for n in reading:
        line1 = n.strip(";\n")
        line1 = line1.replace(":", ";")
        bp_positions_p1.append(line1.split(";"))
with open(reading_file_path2, 'r') as reading2:
    for n in reading2:
        line2 = n.strip(";\n")
        line2 = line2.replace(":", ";")
        bp_positions_p2.append(line2.split(";"))

first_ships = []
second_ships = []
column_counter = 0
row_counter = 0
for line in p1_ships:
    row_counter += 1
    for cell in line:
        column_counter += 1
        if cell == "C" or cell == "S" or cell == "D":
            var = cell + "," + str(row_counter) + chr(column_counter + 64)
```

```
            first_ships.append(var)
    column_counter = 0
row_counter = 0
for line in p2_ships:
    row_counter += 1
    for cell in line:
        column_counter += 1
        if cell == "C" or cell == "S" or cell == "D":
            var = cell + "," + str(row_counter) + chr(column_counter + 64)
            second_ships.append(var)
    column_counter = 0
for ship in bp_positions_p1:
    ship_type = ship[0]
    start_point = ship[1].split(",")
    letter = start_point[1]
    num = start_point[0]
    if ship[2] == 'down':
        if ship_type[0] == 'B':
            first_ships.extend([ship_type + "," + str(x) + letter for x in
range(int(num), int(num) + 4)])
        elif ship_type[0] == 'P':
            first_ships.extend([ship_type + "," + str(x) + letter for x in
range(int(num), int(num) + 2)])
    elif ship[2] == 'right':
        if ship_type[0] == 'B':
            first_ships.extend([ship_type + "," + num + chr(x + 64) for x
in range(ord(letter) - 64, ord(letter) - 60)])
        elif ship_type[0] == 'P':
            first_ships.extend([ship_type + "," + num + chr(x + 64) for x
in range(ord(letter) - 64, ord(letter) - 62)])
for ship in bp_positions_p2:
    ship_type = ship[0]
    start_point = ship[1].split(",")
    letter = start_point[1]
    num = start_point[0]
    if ship[2] == 'down':
        if ship_type[0] == 'B':
            second_ships.extend([ship_type + "," + str(x) + letter for x in
range(int(num), int(num) + 4)])
        elif ship_type[0] == 'P':
            second_ships.extend([ship_type + "," + str(x) + letter for x in
range(int(num), int(num) + 2)])
    elif ship[2] == 'right':
        if ship_type[0] == 'B':
            second_ships.extend([ship_type + "," + num + chr(x + 64) for x
in range(ord(letter) - 64, ord(letter) - 60)])
        elif ship_type[0] == 'P':
            second_ships.extend([ship_type + "," + num + chr(x + 64) for x
in range(ord(letter) - 64, ord(letter) - 62)])


def move_checker(x):
    for move in x:
        try:
            if len(move) < 3:
                raise IndexError
            if type(int(move[:-2])) != int or type(move[-1]) != str:
                raise ValueError
            if int(move[:-2]) > 10 or ord(move[-1]) > 74:
                raise AssertionError
        except IndexError:
```

```
            write("IndexError: Move value {} is
insufficient.\n".format(move))
            moves_p1.remove(move)
        except ValueError:
            write("ValueError: Move value {} is invalid.\n".format(move))
            moves_p1.remove(move)
        except AssertionError:
            write("AssertionError: Move value {} is not
possible.\n".format(move))
            moves_p1.remove(move)


move_checker(moves_p1)
move_checker(moves_p2)
sunk1 = []
sunk2 = []
c = "-"
sub = "-"
d = "-"
p = " - - - -"
b = "- -"
c2 = "-"
sub2 = "-"
d2 = "-"
p_2 = " - - - -"
b_2 = "- -"


def sunk_ship_checker():
    c_counter = 0
    s_counter = 0
    d_counter = 0
    general_p_counter = 0
    general_b_counter = 0
    b1_counter = 0
    b2_counter = 0
    p1_counter = 0
    p2_counter = 0
    p3_counter = 0
    p4_counter = 0
    for element in sunk1:
        element1 = element.split(",")
        type_ship = element1[0]
        if type_ship == "C":
            c_counter += 1
        if type_ship == "S":
            s_counter += 1
        if type_ship == "D":
            d_counter += 1
        if type_ship == "B1":
            b1_counter += 1
        if type_ship == "B2":
            b2_counter += 1
        if type_ship == "P1":
            p1_counter += 1
        if type_ship == "P2":
            p2_counter += 1
        if type_ship == "P3":
            p3_counter += 1
        if type_ship == "P4":
            p4_counter += 1
```

```
    if c_counter == 5:
        global c
        c = "X"
    if s_counter == 3:
        global sub
        sub = "X"
    if d_counter == 3:
        global d
        d = "X"
    if p1_counter == 2:
        general_p_counter += 1
    if p2_counter == 2:
        general_p_counter += 1
    if p3_counter == 2:
        general_p_counter += 1
    if p4_counter == 2:
        general_p_counter += 1
    if general_p_counter == 4:
        global p
        p = " X X X X"
    elif general_p_counter == 3:
        p = " X X X -"
    elif general_p_counter == 2:
        p = " X X - -"
    elif general_p_counter == 1:
        p = " X - - -"
    if b1_counter == 4:
        general_b_counter += 1
    if b2_counter == 4:
        general_b_counter += 1
    if general_b_counter == 2:
        global b
        b = "X X"
    elif general_b_counter == 1:
        b = "X -"
    c_counter2 = 0
    s_counter2 = 0
    d_counter2 = 0
    b1_counter2 = 0
    b2_counter2 = 0
    p1_counter2 = 0
    p2_counter2 = 0
    p3_counter2 = 0
    p4_counter2 = 0
    general_p_counter2 = 0
    general_b_counter2 = 0
    for element in sunk2:
        element1 = element.split(",")
        type_ship = element1[0]
        if type_ship == "C":
            c_counter2 += 1
        if type_ship == "S":
            s_counter2 += 1
        if type_ship == "D":
            d_counter2 += 1
        if type_ship == "B1":
            b1_counter2 += 1
        if type_ship == "B2":
            b2_counter2 += 1
        if type_ship == "P1":
            p1_counter2 += 1
```

```
        if type_ship == "P2":
            p2_counter2 += 1
        if type_ship == "P3":
            p3_counter2 += 1
        if type_ship == "P4":
            p4_counter2 += 1
    if c_counter2 == 5:
        global c2
        c2 = "X"
    if s_counter2 == 3:
        global sub2
        sub2 = "X"
    if d_counter2 == 3:
        global d2
        d2 = "X"
    if p1_counter2 == 2:
        general_p_counter2 += 1
    if p2_counter2 == 2:
        general_p_counter2 += 1
    if p3_counter2 == 2:
        general_p_counter2 += 1
    if p4_counter2 == 2:
        general_p_counter2 += 1
    if general_p_counter2 == 4:
        global p_2
        p_2 = " X X X X"
    elif general_p_counter2 == 3:
        p_2 = " X X X -"
    elif general_p_counter2 == 2:
        p_2 = " X X - -"
    elif general_p_counter2 == 1:
        p_2 = " X - - -"
    if b1_counter2 == 4:
        general_b_counter2 += 1
    if b2_counter2 == 4:
        general_b_counter2 += 1
    if general_b_counter2 == 2:
        global b_2
        b_2 = "X X"
    elif general_b_counter2 == 1:
        b_2 = "X -"


check = False


def hidden_board():
    sunk_ship_checker()
    global game
    global check
    if c == "X" and sub == "X" and d == "X" and p == " X X X X" and b == "X
X" and c2 == "X" and sub2 == "X" and d2 == "X" and p_2 == " X X X X" and
b_2 == "X X" and not check:
        write("It is a Draw!\n\nFinal Information\n\n")
        check = True
        game = "over"
        return
    if c == "X" and sub == "X" and d == "X" and p == " X X X X" and b == "X
X" and not check:
        write("Player2 Wins!\n\nFinal Information\n\n")
        check = True
```

```
        game = "over"
        return
    if c2 == "X" and sub2 == "X" and d2 == "X" and p_2 == " X X X X" and
b_2 == "X X" and not check:
        write("Player1 Wins!\n\nFinal Information\n\n")
        check = True
        game = "over"
        return
    write("Player1's Hidden Board\t\tPlayer2's Hidden Board\n  A B C D E F
G H I J\t\t  A B C D E F G H I J\n")
    check = False
    if tarantino:
        a = round_num
        if round_num == len(moves_p1) + 1:
            move = moves_p2[round_num - 2]
        else:
            move = moves_p2[round_num - 1]
    else:
        a = round_num - 1
        if round_num == len(moves_p1) + 1:
            move = moves_p2[round_num - 2]
        else:
            move = moves_p2[round_num - 1]
    for row in range(10):
        write(str(row + 1))
        for column in range(10):
            x = False
            for number in range(round_num - 1):
                let = moves_p2[number][-1]
                num1 = moves_p2[number][:-2]
                if row == 9 and column + 1 == ord(let) - 64 and row + 1 ==
int(num1):
                    for s in first_ships:
                        s1 = s.split(",")
                        position = s1[1]
                        if position[-1] == let and position[:-1] == num1:
                            check = True
                            if s not in sunk1:
                                sunk1.append(s)
                    if check:
                        write("X ")
                        x = True
                        check = False
                    else:
                        write("O ")
                        x = True
                elif column + 1 == ord(let) - 64 and row + 1 == int(num1):
                    for s in first_ships:
                        if s[-1] == let and s[-2] == num1:
                            check = True
                            if s not in sunk1:
                                sunk1.append(s)
                    if check:
                        write(" X")
                        x = True
                        check = False
                    else:
                        write(" O")
                        x = True
            if row == 9 and not x:
                write("- ")
```

```
            elif not x:
                write(" -")
        write("\t\t")
        write(str(row + 1))
        for column in range(10):
            y = False
            for number in range(a):
                let2 = moves_p1[number][-1]
                num2 = moves_p1[number][:-2]
                if row == 9 and column + 1 == ord(let2) - 64 and row + 1 ==
int(num2):
                    for s in second_ships:
                        s1 = s.split(",")
                        position = s1[1]
                        if position[-1] == let2 and position[:-1] == num2:
                            check = True
                            if s not in sunk2:
                                sunk2.append(s)
                    if check:
                        write("X ")
                        y = True
                        check = False
                    else:
                        write("O ")
                        y = True
                elif column + 1 == ord(let2) - 64 and row + 1 == int(num2):
                    for s in second_ships:
                        if s[-1] == let2 and s[-2] == num2:
                            check = True
                            if s not in sunk2:
                                sunk2.append(s)
                    if check:
                        write(" X")
                        y = True
                        check = False
                    else:
                        write(" O")
                        y = True
            if row == 9 and not y:
                write("- ")
            elif not y:
                write(" -")
        write("\n")

write("\nCarrier\t\t{}\t\t\t\tCarrier\t\t{}\nBattleship\t{}\t\t\t\tBattlesh
ip\t{}\nDestroyer\t{}\t\t\t\t"
          "Destroyer\t{}\nSubmarine\t{}\t\t\t\tSubmarine\t{}\nPatrol
Boat{}\t\t\tPatrol Boat{}\n\n"
          .format(c, c2, b, b_2, d, d2, sub, sub2, p, p_2))
    if game != "over":
        write("Enter your move: {}\n\n".format(move))


write("Battle of Ships Game\n\n")
for round_num in range(1, len(moves_p1) + 2):
    if game != "over":
        write("Player1's Move\n\nRound : {}\t\t\t\t\tGrid Size:
10x10\n\n".format(round_num))
        tarantino = False
        hidden_board()
        tarantino = True
```

```
        write("Player2's Move\n\nRound : {}\t\t\t\t\tGrid Size:
10x10\n\n".format(round_num))
        hidden_board()
if game == "over":
    check = True
    round_num = len(moves_p1)
    hidden_board()
```

4.2 Time Spent on This Assignment

Time spent on analysis	During the analysis of this assignment, I read the pdf file and all the attachments thoroughly and this took approximately 2 hours.
Time spent on design and implementation	After understanding the problem, designing and implementing a solution was the part that took the most time. 15 hours were approximately how much this section took.
Time spent on testing and reporting	Since there are many things that can go wrong with the scenario, I tested so many different moves, ship positions, different winners etc. to see if my code was working properly. In this step, there were some errors I detected and fixing these with the whole testing part took around 10 hours. The reporting part took about 3 hours.

4.3 Reusability

While writing my code I tried to put relevant variable names and defined my functions with names that correspond with what they do when called. Therefore, I believe that my code is reusable.

5 User Catalogue

In this section, how to use my Battle of Ships code to play the game is explained to users.

Ship Placements:

Players each need to put their ship positions in txt files. This file would look like this.

```
;;;;;;C;;;
;;;;B;;C;;;
;P;;;B;;C;P;P;
;P;;;B;;C;;;
;;;;B;;C;;;
;B;B;B;B;;;;;
;;;;;S;S;S;;
```

```
;;;;;;;;;D
;;;;P;P;;;;D
;P;P;;;;;;;D
```

There are multiple battleships and patrol ships. When placed next to each other, determining which P letter belongs to which patrol ship causes uncertainty. This problem is solved by the players adding another txt file called OptionalPlayer1.txt for the first player and OptionalPlayer2.txt for the second. This file would look like this.

```
B1:6,B;right;
B2:2,E;down;
P1:3,B;down;
P2:10,B;right;
P3:9,E;right;
P4:3,H;right;
```

Player Moves:

Each player's move for each round is decided beforehand. These moves are in txt files and a file's content containing three moves would look like this: "5,E;10,G;8,I;"

Possible Errors:

There are many errors that might be faced during the gameplay. If players enter invalid moves like ",;, / B,C; / 15,X;" these moves are removed from the file right after informing you of the situation. If there is a problem with the files' placement, name etc. the game doesn't work and informs you of the problem. If any other problem occurs, you will be notified by this message: "kaBOOM: run for your life!".

Output System:

In each round; round number, whose round it is, current situation of the players' boards, current situation of their ships and the next player's move is given. This information is given to the users via command line and also the output file that will be taken as argument such as "Battleship.out".

6 Grading Table

Readable Codes and Meaningful Naming	5
Using Explanatory Comments	5
Efficiency (avoiding unnecessary actions)	2
Function Usage	15
Correctness, File I/O	25
Exceptions	15
Report	20